15418-finalproj

Title

Parallelized Path Planning Algorithms, by Aidan Lincke (alincke) and Soren Dupont (sdupont)

URL

https://aidanlincke.github.io/15418-finalproj/

Summary

Our plan for this project is to implement three path planning algorithms (Dijkstra's algorithm, PRMs, and RRT-Connect) on graphs that have some level of parallelism that can be exploited, but that are difficult to extract, and compare which algorithms best lend themselves to parallel speedup. We will use OpenMP to program familiar graph search algorithms such as Dijkstra's algorithm and RRT-Connect. Probabilistic roadmaps, which involve a higher degree of parallelism for generating random points, may be well-suited for parallelism via CUDA. In 15-210, we learned that it should be possible to run an algorithm within S + W/P time, where S is span, W is work, and P is the number of processors. Is this time bound practically achievable?

Background

Path planning algorithms are used in many fields ranging from robotics to navigation. Because of the plethora of applications of path planning, many algorithms tuned for different situations exist, such as Djikstra's algorithm, PRM planners, and RRT-Connect. However, the problem of path planning can quickly become computationally difficult. For example, as the number of dimensions of state grows, search graphs quickly become large and highly-connected. In addition, relatively optimal paths need to be determined in reasonable time in many applications such as walking or driving directions. Parallelizing these path-planning algorithms can offer the opportunity to find more optimal paths in less time.

The Challenge

This problem is challenging because while there is some parallelism in graph path-finding algorithms that can be exploited, there are many sensitive shared resources that processors must have access to in order to make progress, such as the graph itself. For example, Dijkstra's algorithm has a shared set of visited nodes, a priority queue for which node should be accessed next, and a hashmap containing the shared graph itself. Allowing processors to work with slightly stale versions of these resources could be a solution, as long as each processor maintains consistent versions of those resources within itself. Similarly, we may explore changing how a resource like the priority queue is accessed (for example, perhaps elements of it could be assigned to certain nodes and sent asynchronously to reduce contention over the priority queue). This algorithm introduces a complex set of tradeoffs that may need to be considered and tested for.

In terms of finding approaches to add parallelism to Djikstra's, we'll also explore partitioning the search graph. If contention for shared resources prohibits speedup as previously mentioned, then we could remove the shared resources by partitioning the graph into sections with start and end points, solving Djikstra's on each section, and then piecing the sections together. However, determining computationally-equal

partitions with start and end points that can be stitched together is a difficult task. In our initial considerations, we believe that the cost of computation on Dijkstra's varies with connectivity of the graph, so we hope to divide the graph into partitions of similar connectivity. We are thus considering certain precomputations that might be done to help with load balancing in this case. Graph contraction techniques could play a role in this, and that itself is a difficult parallelizable task, as it involves many nodes reading from and modifying a graph concurrently.

Parallelism in probabilistic roadmaps lends itself much more to CUDA kernels. For example, during the generation phase of a probabilistic roadmap planner, millions of points are constructed and connected to their nearby neighbors within certain rules. With CUDA, these random points could be generated an order of magnitude faster, enabling better roadmaps, lower-cost results, and a quicker path. However, difficulties could be foreseen with this CUDA-based approach such as certain points not having any neighbors and thus completing much quicker than points potentially containing many neighbors.

For each of these algorithms, there is a certain amount of spatial locality in the search graphs and the shared resources. However, cache contention over these resources could severely hamper performance. In addition, access patterns could also prove to be challenging because graphs are likely to follow non-ideal access patterns during the exploration of edges.

Resources

We'll use the GHC and PSC machines to run, debug, and test the performance of our code. We'll use Wikipedia and other resources for the search algorithms of Dijkstra's, PRMs, and RRT-Connect. We may also use code from the CMU class 16-350 Planning Techniques for Robotics, containing only sequential implementations of these path-planning algorithms because the class does not discuss parallelism.

Goals and Deliverables

We plan to implement three path finding algorithms for graphs that have parallelism that can be exploited (Dijkstra's, Probabilistic Roadmap Planning, and RRT-Connect). We plan to optimize our implementations of these algorithms, compare the difficulties of unlocking parallelism in these algorithms, and describe the performance gain achievable for each of them.

We hope to explore more niche graph search / path planning methods and how they may be used to assist in improving performance of these algorithms. For example, can precomputation / helper procedures / programs such as graph contraction used to improve load balancing or provide heuristics that may sacrifice quality of answer for better performance?

We also hope to compare theoretical limits on how much execution time could possibly be reduced for these algorithms, with what we are able to achieve. This would be difficult to do as work and span analyses typically ignore constant values, so additional research would be required to get exact values for execution time of certain operations so that a comparison can be done with observed execution time.

Platform Choice

We have chosen parallelizing with OpenMP on shared address space machines such as the Gates and PSC machines. A shared address space model is well suited to parallelizing graph pathfinding algorithms because of the high amount of shared data between processors involved in these algorithms, such as the graph itself, and data structures storing which nodes still have to be explored, paths already found, etc.

Staleness could be very problematic for executing these algorithms and still having quality answers, but a shared address space would also enable some amount of staleness being introduced if it is worth a tradeoff in performance by reducing contention over these shared resources.

Because graph search algorithms are generally not highly parallel, and graphs themselves are typically highly irregular, GPU/CUDA programming is unlikely to be well suited for most of these algorithms. However, we would like to see if it makes sense to utilize them for the Probabilistic Roadmaps algorithm, which involves generating a large amount of random points in a highly parallel fashion.

Schedule

Dates	Objective
3/26 - 4/2	Create path-planning problem with map. Plan out how we will implement which algorithms; do deeper research on them, our options, opportunities for parallelism, approach that can be taken, etc.
4/2 - 4/9	Complete a prototype of Dijksta's Algorithm that can be easily modified to try different approaches of exploiting parallelism / different policies that might yield differing results.
4/9 - 4/16	Finalize getting the best speedup possible for the first algorithm and record results. Begin implementing two more (PRMs and RRT).
4/16 - 4/23	Finalize these two algorithms and record results.
4/23 - end	Depends on where we stand at this point. There are a few niche things related to this topic that we are interested in exploring that we may have or may have not already looked at at this point (for example seeing if CUDA can be used, utilizing and parallelizing helper-procedures such as graph contraction, and whatever else may come up!); compare our results with theoretical speedup for these algorithms. Finalize results and poster board.